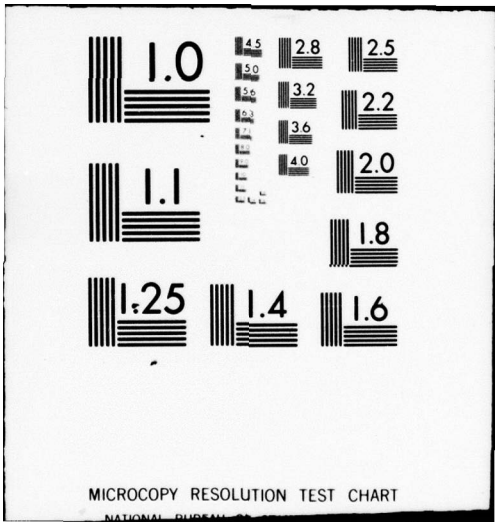
AD-A070 801 SYSTEM DEVELOPMENT CORP SANTA MONICA CALIF F/G 5/2
DEDUCTIVE PLANNING AND PATHFINDING FOR RELATIONAL DATA BASES,(U)
1978 C KELLOGG, P KLAHR, L TRAVIS N00014-76-C-0885
UNCLASSIFIED NL

| OF |
AD
A070801

END
DATE
FILMED
8-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART

FROM LOGIC & DATA BASES, HERVÉ GALLAIRE AND JACK MINKER, EDS., PLENUM PRESS, N.Y. 1978.

Contract N00014-76-C-0885

DATE 1978

DEDUCTIVE PLANNING AND PATHFINDING FOR RELATIONAL DATA BASES

Charles Kellogg[1], Philip Klahr[2], and Larry Travis[2]

System Development Corporation, Santa Monica, California[1]

University of Wisconsin, Madison, Wisconsin[2]

ABSTRACT

Inference planning techniques have been implemented and incorporated within a prototype deductive processor designed to support the extraction of information implied by, but not explicitly included in, the contents of a relationally structured data base. Deductive pathfinding and inference planning are used to select small sets of relevant premises and to construct skeletal derivations. When these "skeletons" are verified, the system uses them as plans to create data-base access strategies that guide the retrieval of data values, to assemble answers to user requests, and to produce proofs supporting those answers. Several examples are presented to illustrate the current capability of the prototype Deductively Augmented Data Management (DADM) system.

INTRODUCTION

Not only are computerized data bases growing in size, number, and complexity, but the number of on-line users is also growing rapidly. The availability of larger and cheaper memories is making it feasible to store vast quantities of data on-line, but this often serves only to increase the frustration of users, who, because of limitations in current data-base retrieval technology, are unable to take full advantage of the information. A major deficiency in present data-base systems is an inability to discover (at the direction of users) implicit relationships among the data items explicitly present.

Deductive logic offers considerable potential for improving on-line access to large, complex data-base domains. The prototype Deductively Augmented Data Management (DADM) system described in this paper has been designed to:

1. Permit a user to pose complex and subtle queries to the system, which, in turn, finds inferential connections linking user-specified concepts to data-base structures.

2. Generate for the user deductively connected evidence chains that he can use in evaluating the utility and credibility of information derived from the data base.*

In particular, user-system interactive techniques have been developed whereby the system creates and displays inference plans and chains of evidence as an integral part of the question-answering process. The user actively participates by supplying advice, refining his queries, and requesting additional plans and evidence as necessary. This interactive cycle continues until the user is satisfied with the quality as well as the quantity of the derived information. Sometimes this entails the provision of evidence both for and against a user's conjecture or working hypothesis. Sometimes the system provides a user with a conditional (yes if...) answer rather than a strictly categorical answer. In all cases, the system permits a user to ask for corroborative evidence by requesting alternative derivations for an answer. (Multiple evidence chains may often reinforce the user's confidence in the value of the information received.)

APPROACH

The design for the deductive processor described in this paper evolved out of research on an English question-answering system called CONVERSE (Kellogg et al. [1971] and Travis et al. [1973]). This system consisted of a language processor (driven by English syntax rules and a semantic network) and a relational data management system that accessed specific facts realized as N-tuple members of predicate (relation) extensions. When analyzing a query such as "Who is mayor of Denver?", the system would use its semantic network to infer that the reference was to the City of Denver, not the County of Denver. The inference was based on the general proposition, represented in the semantic network, that the range of the relation being mayor of includes cities but not counties.

* It is important to note that while the deductive processor will be applying rules of strict logical reasoning, the information (the set of general assertions or premises) that is being used to construct evidence chains may range in degree of plausibility from "hard" (strictly true) to "soft" (possibly the cause).

Further, in analyzing more complex queries such as "What cities are in states with a population less than that of the City of Boston?", the system would infer which states possess the property of having a population smaller than that of Boston--an ad hoc property not directly available in the network or data base. While useful, these kinds of inferences are special purpose and limited. We decided that a more general-purpose inferential capability needed to be designed and added to the system for use in many different contexts and for many different purposes (Klahr [1975], Kellogg et al. [1976], Klahr [1978], and Kellogg et al. [1977]).

Two design criteria were crucial in the development of the deductive processor (DP). The first criterion was that the DP would be an independent system yet capable of being "added on" to existing and emerging relational data management systems (RDMSs). This led to a distinct separation between a store of extensional data (specific facts) and a store of intensional data (general statements, premises, rules). The former is accessed by an RDMS, while the latter is accessed by the DP (see Figure 1). (This separation of data is also suggested in a recent proposal by Reiter [1978].) No change is necessary to the RDMS to add on the DP. This same criterion of an RDMS add-on also led to a focus on deduction by exception: user queries not requiring deduction should be identified as such and sent directly to the RDMS.

The second criterion focused on the selection of relevant premises. Premises, or inference rules, are general statements that can be used in making deductions. Given a large number of such premises, a crucial problem arises in controlling the deductive search space. An inference planning process has been designed and implemented to locate potentially relevant premises. This process must be fast and efficient to compensate for the overhead processing involved. But such planning is needed in order to give the system guidance in its deductive searching. Furthermore, the planning process is used to guide and direct relational data-base searching by specifying what facts are needed to support the deductions and proofs found to answer user queries.

ABSTRACTING AND SEMANTICALLY RESTRICTING DEDUCTIVE INTERACTIONS

Processes of abstraction (of deductive interactions) and restriction (of semantic scope) are central to our approach to relevant premise selection. Where possible these abstraction and restriction processes are carried out during premise input in order to minimize processing time during query analysis and deductive question-answering.

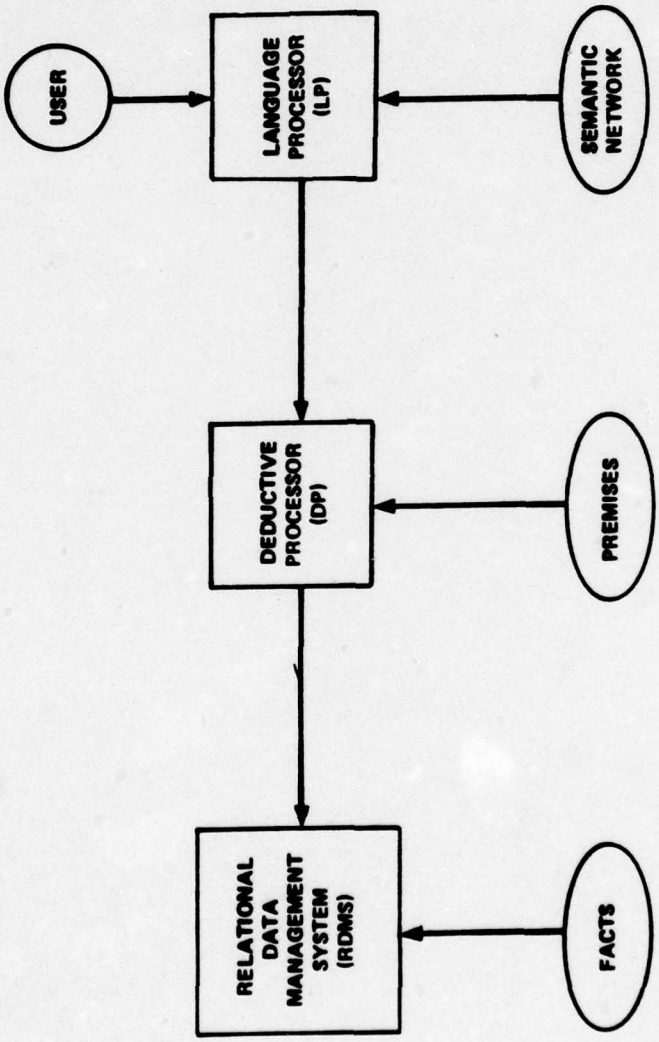


Figure 1. Addition of a Deductive Processor to an English Question-Answering System

Premises and queries are entered into the system as primitive conditional statements (Travis et al. [1973]).* A primitive conditional is a first-order predicate-calculus normal form whose central connective is the implication sign. The antecedent of the implication contains the assumptions of the premise/query and the consequent contains the goals of the premise/query. Assumptions and goals are literals, that is, atomic predicate occurrences or negated atomic predicate occurrences. Within a given antecedent or consequent, literals may be combined either conjunctively or disjunctively. Each predicate occurrence is an instance of a predicate (relation) along with its argument terms (namely variables, constants, or functions). Primitive conditionals are used because they support the introduction of general assertions in a natural way, similar to the way production rules are used in knowledge-based systems; see Davis and King [1975].

Several kinds of information are abstracted from the premises during input and used to create a predicate connection graph (PCG),** as well as other storage structures that promote efficient association of deductive and semantic information (Klahr [1975]). A premise is first converted into a Skolemized, quantifier-free form. The implication (as well as other truth-functional) connections among the predicate occurrences in a premise are encoded into the PCG as a series of deductive dependency Links. Further, the deductive interactions (or unifications--see Robinson [1965]) between predicate occurrences in the new premise and predicate occurrences in existing premises are pre-computed and encoded into the PCG as a series of interpremise associative Arcs. The variable substitutions required for unification are stored elsewhere, for later use in verifying skeletal derivations (i.e., inference or proof) plans.

Semantically restrictive information is introduced in several different forms in order to restrict the logically possible unifications to those that are semantically meaningful for particular application domains.

The variables and constants occurring in premises can be "typed", that is, assigned to specific domain classes. For example, the variable "X" might be assigned the type DOCUMENT, and the constant "Sam" assigned the type SCIENTIST. Then, whenever "X" and "Sam" occur in the same argument position of different instances of a relation, those relation instances will not unify, and they will not be connected in the PCG, due to their semantically

* In an operational system, premises would normally be entered by the data-base administrator.

** See Kowalski [1975] and Sickel [1976] for the use of connection graphs in theorem proving.

incompatible types.

Compound types, consisting of set union, intersection, and difference operations over simple types, may also be used to specify more complex semantic restrictions on predicate domains. A semantic network is used to represent set relationships between types.* Class inclusion paths within this network are used, for example, to permit unification of instances of type SCIENTIST with instances of type MAMMAL. As new premises are entered into the system, this semantic network is automatically updated to reflect new predicate-domain associations.

In addition to this use of semantic information to restrict unification by means of types, unification between multiple occurrences of a predicate within the same premise may sometimes be avoided by restating the premise's assertion by use of logical properties. For example, the predicate "North-of" could be characterized by the premises:

$$\forall x \forall y \ (\text{North-of}(x,y) \ \& \ \text{North-of}(y,z) \supset \text{North-of}(x,z))$$

$$\forall x \forall y \ (\text{North-of}(x,y) \supset \neg \text{North-of}(y,x))$$

$$\forall x \quad (\neg \text{North-of}(x,x))$$

The first premise specifies that North-of is transitive. This premise is recursive and can deductively interact with itself and the other premises to cause a rapid expansion of the deductive search space. To help avoid this problem, the DADM system permits binary predicates to be characterized by their logical properties (for example North-of would be assigned the logical properties: transitive, asymmetric, and irreflexive). Computational procedures can then be called to effect special-purpose inferences associated with various groupings of logical properties. Recursive premises describing logical properties of predicates are therefore replaced, where possible, by special-purpose subroutines. Subroutines are being implemented for consistent combinations of the logical properties identified by Elliott [1965].** Future effort will involve other properties such as a relation being hereditary with respect to another relation, e.g., P being hereditary over R in

$$\forall x \forall y \ P(x) \ \& \ R(x,y) \supset P(y)$$

---

* See McSkimin and Minker [1977, 1978] for related research on introducing semantic information into a deductive system.

** Properties and examples are: reflexive (equal-to), irreflexive (greater-then), symmetric (equal-to), asymmetric (North-of), transitive (located-in), 1-leader (mother-of), 1-follower (weighs), noregrowth (son-of), and unlooped (mother-of).

Logical properties of binary relations are identified by a user-system dialog that is initiated, as shown below, for the predicate "North-of" (user input is preceded by an asterisk):

* Define (North-of)

Suppose one thing is North-of a second thing that in turn is North-of a third thing. Is the first thing North-of the third?

* Yes

If one thing is North-of a second thing, will it always be the case that the second is North-of the first?

* No

Might it ever be the case?

* No

After the third yes/no response, the system is able to identify "North-of" as a transitive, asymmetric, irreflexive, and unlooped relation.

Variable typing reduces the number of unifications in the PCG by making use of semantic domain restrictions. Logical properties replace some kinds of recursive premises, and their often troublesome unifications, with special-purpose inferencing procedures. A third form of semantic restriction used in the DADM system does not directly eliminate unifications in the PCG, but does limit the selection and use of premises and predicates by means of advice supplied by a data-base administrator or user during query processing.

A data-base administrator enters semantic advice in the form of "Conditions → Recommendations" rules. For example, one could advise that a ship return to its home port if it is damaged by specifying:

(Assumption Damaged(Ship)) → Returns(Ship Ports)

The system would try using premises containing the Returns relation when the Damaged relation occurs as an assumption. Advice rules are stored in an advice file, where they are automatically selected and applied whenever their condition part holds for input queries. In addition to such advice rules, the user could supply advice for a particular query by stating only the advised recommendation for that query.

Advice most typically involves recommendations on the use of particular premises or predicates in finding deductions. For advised premises, the system will try using them whenever possible in the course of constructing a proof. For advised predicates, the system will try chaining through occurrences of them in premises. In the case of negative advice, specified premises and predicates are avoided in proof construction.

INFERENCE PLANNING AND DEDUCTIVE QUESTION ANSWERING

The development, refinement, and execution of inference plans proceeds through a series of phases. These phases are designed to progressively apply a series of increasingly more stringent deductive, semantic, and pragmatic constraints until a user receives his desired information or is convinced that he has explored all reasonable deductive pathways into the data base. These phases are described below.

Deductive Pathfinding

Symbolic queries (in the form of primitive conditionals) are decomposed into a set of assumptions (antecedents of the conditional) and a set of goals (consequents of the conditional). Deductive pathfinding employs a process of middle-term chaining (Klahr [1978]) to be illustrated later. This process uses the predicate connection graph to find chains of middle-term predicates needed to deductively connect assumptions to goals. Middle-term chaining combines the processes of forward chaining from the assumptions in a query and backward chaining from the goals in a query. When a query contains no assumptions, and the system cannot discover plausible ones to use--say, as a result of semantic advice--middle-term chaining defaults to backward chaining. As chaining proceeds, a series of expanding deductive-interaction "wave fronts" are generated from assumptions toward goals and from goals toward assumptions. Intersections are performed on the wave fronts until a non-empty intersection occurs, at which time the system has found an implication chain from an assumption to a goal. Several such implication chains are usually found (shortest chains first) before a user-controlled limit is reached. Middle-term chaining is further constrained by the use of semantic advice and plausibility measures. The plausibility measures are assigned to premises and are used to order the predicate occurrences comprising middle-term chain wave fronts to ensure that the deductive paths involving the most plausible premises are selected first. In a similar fashion, semantic advice obtained from the advice file or from the user is transformed into premise and predicate alert lists that are used to ensure that advised premises and predicates are given priority or avoided, depending upon whether the advice is positive or negative.

The same assumptions may be used to find deductive support for different goals (and subgoals). When assumptions are not supplied in a query, useful assumptions may sometimes be found by following semantic network predicate-domain connections, or by using advised predicates as possible assumptions.

Plan Generation

For each middle-term chain generated, the system extracts the premises whose occurrences are part of the chain. Subgoals resulting from the premises are set up to be resolved either by deductive support through the premises, by data-base search through the relational file, or by procedural computation. Subgoals are added to a proof-proposal tree, which contains the inference plans being formed and developed. Once inference plans have no remaining deductive subgoals, they are available for verification, user review, and instantiation.

Plan Verification

Skeletal plans constructed during plan generation are valid proofs at the truth-functional level. In plan verification, the variable substitutions associated with the unifications in each plan are examined for consistency. If there are no clashes--that is, if no variables are assigned more than one distinct constant value--then verification is successful and instantiation by data-base search may follow. During this stage, classes of variables that must take on the same value are constructed and used to reformulate skeletal derivations into search-compute plan components (i.e., data-base access strategies) and inference plan components (comprising deduced goals, deduced subgoals, and assumptions).

Plan Review, Plan Selection, and Query Refinement

Though on-line interaction may be initiated by the user or prompted by the system at various points during pathfinding and plan generation, most user review and interaction occurs after plan verification. Verified plans are usually reviewed in the order in which they were generated. (Recall that plans using the shortest paths, most plausible premises, and advised premises and predicates are generated first.)

During review, a user may reject a plan, instantiate it (by requesting data-base search) or suspend further action on it until other plans have been reviewed. In this manner, the user can minimize unnecessary data-base searching by reviewing the derived plan information and reaching conclusions about the likely data-base searching consequences of his original request. Plan review may, for example indicate that additional assumptions, goals, or advice should be associated with the original request, or that the original

query should be refined or replaced by a more specific (or general) request. Considerable insight into interpreting complex requests with respect to large data bases can be achieved, short of actually searching the data base, by this process.

Data-Base Search and Answer Generation

An inference plan constitutes a complete proof just in case no search/compute plan is produced (i.e., all subgoals are deduced from premises). More typically, one or more subgoals require data-base and/or procedural (compute) support. Search/compute plans are executed, in general, in three phases: first, all computable functions and predicates having only constants as arguments are evaluated; second, a sequence of relational search requests is executed against the data base; third, remaining computable functions and predicates are applied to the results of data-base search. Answers are extracted from the N-tuples of data values associated with search/compute plan variables. (Each of these N-tuples supplies instantiation values that may be used to convert the original inference plan into a complete proof or "chain of evidence".) An answer may be categorical (for example, "yes" if no variables occur in the original request, and data-base search is satisfied), descriptive (a set of search-derived query-variable values displayed in tabular format), or conditional ("yes if..." the specified predicate-argument conditions can be verified by the user to hold true for the application domain).

Often these categorical, descriptive, or conditional answers will satisfy the user's original information requirement. In other cases, he may wish to proceed to the next (and final) step in the inference plan development-execution-review cycle.

Answer Explanation and Evidence Review

Just as the plan review, plan selection, and query refinement process is designed to aid the user in understanding the full computer-developed implications of his query, the answer explanation and evidence review phase of processing is designed to support him in his evaluation of computer-derived answers. In a later section, several computer examples illustrate current proof displays. Though this form is often sufficient to enable users to determine the validity and/or utility of derived answers, a more interactive and easily comprehended dialog format for evidence display is under development. This new facility will permit a user to selectively interrogate the system concerning particular answers, relations, and domains. By repetitive interrogation, he may delve as deeply as he desires into particular lines of reasoning or evidentary support, without resorting to the current practice of full proof display.

Inference Planning, Data-Base Semantics, and Generalized Navigation

The relational (extensional) data base constitutes a logical model or interpretation for many of the relations used in the premise (intensional) file. Conversely, the intensional information constitutes a partial but precise representation of the semantics of the extensional data base. Inference planning uses this intensional information to develop both the semantic implications of user-request assumptions and the semantic antecedents of user-request goals. Therefore, inference planning may be used to support generalized navigation or browsing operations through the semantics of a data base. Generalized navigation is further supported by allowing users to enter requests containing unrestricted relations (i.e., relations with no arguments). Given queries of this sort, the system can quickly find deductive paths through system restricted concepts supporting goal relations and concepts linking assumptions to goals. This system feature has proved most useful as a tool for exploring the interrelationships between intensional concepts.

DEDUCTIVE PROCESSOR COMPONENTS

Figure 2 shows the components of our DADM system prototype. At present, users communicate directly with the control processor; a language processor will be incorporated at a later date. The control processor accepts premises and queries in primitive conditional form as well as user advice and commands. It accesses and coordinates the use of the several system components briefly described below.

Array Initialization and Maintenance

Information abstracted from the premises is segmented into seven internal arrays. This segmentation contributes to system modularization and increases processing efficiency. The seven arrays are:

(1) Premise Array. Each entry represents a premise and contains a list of the predicate occurrences in the premise, the plausibility of the premise, and the premise itself (both symbolic and English) for purposes of display.

(2) Predicate Array. The predicate array contains the relations known to the system as well as the support indicator associated with each relation, which indicates how to resolve each relation when it occurs as a subgoal (deduce, search data base, compute).

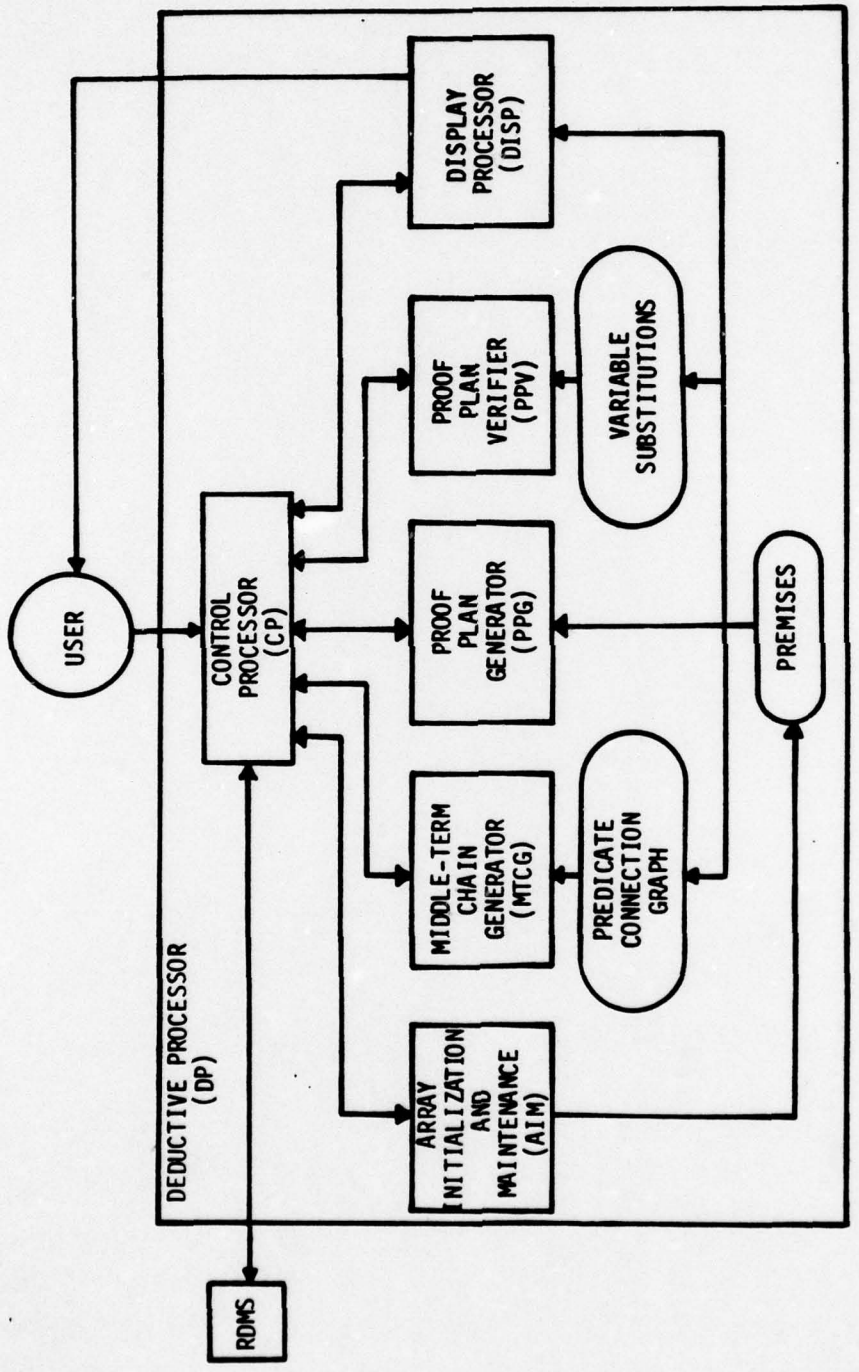


Figure 2. Deductive Processor Components

(3) Predicate Occurrence Array. Each entry represents a predicate occurrence and contains the following information about the occurrence: its predicate name (index into predicate array), the sign of the occurrence (positive or negative), whether the occurrence is in the antecedent or consequent of the implication, the main connective (conjunction or disjunction) governing the occurrence, and the numerical position of the occurence within its premise. The information is compactly stored in a single-word bit vector to save storage space.

(4) Argument Array. The argument strings of the predicate occurrences are stored in the argument array in one-to-one correspondence to the positions of the occurrences in the predicate occurrence array.

(5) Link Array. Truth-functional dependencies within premises are stored in the link array. These dependencies can be implicational, disjunctive, or conjunctive. For each predicate occurrence, a list of the occurrences with which it is truth-functionally connected is entered into the array.

(6) Unifications Array. Each entry contains a list of the unifications (deductive interactions) associated with the given occurrence. The unifications array and the links array comprise the predicate connection graph.

(7) Variable-Substitutions Array. The substitution lists associated with unifications are stored in one-to-one correspondence with the positions of the unifications in the unification array.

Chain Generator, Plan Generator, and Plan Verifier

The Chain Generator, Plan Generator, and Plan Verifier support the deductive pathfinding, plan generation, and verification processes. They communicate with one another by means of the control processor and with the user by means of the display processor.

Display Processor

Plan and proof (evidence) review and query refinement processes are supported by the Display Processor. The user can, for example, examine middle-term chains generated, plans formed, subgoals, verified plans, data-base search requests, data-base values returned, answers, completed proofs, and premises used in proofs.

DEDUCTION EXAMPLES

Figures 3 and 4 illustrate the current operation of the deductive processor (DP) prototype interfaced to a small RDMS. (Both DP and RMS are written in LISP 1.5 and operate on SDC's Amdahl 470/V5 computer.)

The first example illustrates the generation of short inference and search/compute plans for the question, "What ships are closer to the Kittyhawk's home port than the Kittyhawk is?" The query is first shown in English and then in the primitive conditional symbolic form that the prototype currently recognizes. The query is expressed in terms of a conjunctive goal composed of the predicates CLOSER-THAN and HOME-PORT. Constants (such as Kittyhawk) are specified by being enclosed in parentheses, while variables (such as x and y) are not. One of the query goals (HOME-PORT) is to be given data-base support; that is, it has been defined by data base values, while the other goal (CLOSER-THAN) is to be deduced. Since the antecedent in the query is empty, the system back-chains from CLOSER-THAN through premise 29. The plausibility of the plan in this case is simply the plausibility of the single premise used (plausibility measures are assigned by the data-base administrator and range from 1 (very low plausibility) to 99 (always the case)). Two new search requests (in addition to HOME-PORT) result from premise 29, as well as a compute relation containing functional arguments. Computations for the functions and the relation are delayed until values for the variables x and y (the values needed to satisfy the search requests) have been found in the data base.

The system sends the four search requests to the RDMS, which finds two ships, the Forrestal and the Gridley, that are closer to the Kittyhawk's home port (San Diego) than the Kittyhawk is. The system then displays the proof that led to the first answer (the Forrestal). A proof using the other answer would be identical to this one except that Gridley would replace Forrestal in the proof, and the distance between the Gridley and San Diego would replace 310 (the distance between the Forrestal and San Diego). The symbols G2, G3, etc., represent nodes in the proof proposal tree and are used here for reference. G2 and G3 represent the original goals as also shown in the inference plan. G5, G6, and G7 are subgoals that resulted from premise 29, which was used to deduce G2. Thus, these three subgoals are indented below G2.

The middle-term-chaining and planning processes are more evident in the example in Figure 4. The input query contains two assumptions (DAMAGED and DESTINATION) and one goal (TRANSPORT). Taurus and NY are constants; Cargo and x are variables. The query asks the system to find values for x that satisfy the query. The variable x is also restricted to range over ships. This is an example of a type restriction on a variable. In the course of

```
*WHAT SHIPS ARE CLOSER TO THE KITTYHAWK'S HOME PORT
*THAN THE KITTYHAWK IS?

QUERY((()IMP(AND(CLOSER-THAN X (KITTYHAWK) Y)
                 (HOME-PORT (KITTYHAWK) Y))))
INFERENCE PLAN:
   DEDUCE   G2 *CLOSER-THAN X KITTYHAWK Y
   SEARCH   G3 *HOME-PORT KITTYHAWK Y
PREMISES USED: (29)     PLAN PLAUSIBILITY:  99
SEARCH/COMPUTE PLAN:
   SEARCH      *SHIPS KITTYHAWK
   SEARCH      *SHIPS X
   SEARCH      *HOME-PORT KITTYHAWK Y
   COMPUTE     *GREATER-THAN (DISTANCE-BETWEEN KITTYHAWK Y) (
                DISTANCE-BETWEEN X Y)
ENTERING DATA BASE
DATA-BASE SEARCH SUCCESSFUL
***************
ANSWER SUMMARY --
VARIABLES:
(X Y)
ANSWERS:
(FORRESTAL SAN-DIEGO)
(GRIDLEY SAN-DIEGO)
***************
PROOF DISPLAY:
   DEDUCED  G2 *CLOSER-THAN FORRESTAL KITTYHAWK SAN-DIEGO
   FACT     G5 **SHIPS KITTYHAWK
   FACT     G6 **SHIPS FORRESTAL
   COMPUTED G7 **GREATER-THAN 378 310
   FACT     G3 *HOME-PORT KITTYHAWK SAN-DIEGO
PREMISES USED: (29)     PROOF PLAUSIBILITY:  99
TYPE PREMISE NUMBER TO DISPLAY, OR 'END':
29
((ALL X79) (ALL X80) (ALL X81)
 (AND (SHIPS X79) (SHIPS X80))
  (GREATER-THAN (DISTANCE-BETWEEN X79 X81)
   (DISTANCE-BETWEEN X80 X81)))
 IMP (CLOSER-THAN X80 X79 X81))
PLAUSIBILITY:  99
TYPE PREMISE NUMBER TO DISPLAY, OR 'END':
END
END DISPLAY
```

Figure 3. Deduction Involving Deduce, Data-Base Search, and Compute Predicates

```
*IF THE TAURUS WERE DAMAGED WHILE DESTINED FOR NEW
*YORK WITH A CARGO, WHAT SHIPS COULD TRANSPORT THE
*CARGO TO NEW YORK?

QUERY(((WHAT (SHIP . X))
       (AND (DAMAGED (TAURUS))
            (DESTINATION (TAURUS) (NY) CARGO))
       IMP (TRANSPORT X CARGO (NY))))
INFERENCE PLAN:
   DEDUCE   G1 *TRANSPORT SHIP#X X75 NY
   ASSUME      *DESTINATION TAURUS NY X75

   DEDUCE   G3 **OFFLOAD TAURUS X75 X72
   ASSUME      **DAMAGED TAURUS
   MID-TERM    **RETURNS TAURUS X72

PREMISES USED: (23 7 15)     PLAN PLAUSIBILITY:  80
SEARCH/COMPUTE PLAN:
   SEARCH      *HOME-PORT TAURUS X72
   SEARCH      *CARRY TAURUS X75
   SEARCH      *AVAILABLE SHIP#X X72
ENTERING DATA BASE
DATA-BASE SEARCH SUCCESSFUL
***************
ANSWER SUMMARY --
VARIABLES:
(X)
ANSWERS:
(PISCES)
(GEMINI)
***************
PROOF DISPLAY:
   DEDUCED  G1 *TRANSPORT PISCES OIL NY
   ASSUME      *DESTINATION TAURUS NY OIL

   DEDUCED  G3 **OFFLOAD TAURUS OIL FREEPORT
   ASSUME      **DAMAGED TAURUS
   MID-TERM    **RETURNS TAURUS FREEPORT

   FACT     G11***HOME-PORT TAURUS FREEPORT
   FACT     G12***CARRY TAURUS OIL
   FACT     G4 ***AVAILABLE PISCES FREEPORT
PREMISES USED: (23 7 15)      PROOF PLAUSIBILITY:  80
END DISPLAY
```

Figure 4. Deduction Using Middle-Term Chaining

developing deductions, the system will not allow values that belong to domain classes other than ships to be substituted for x.

The inference plan shown in Figure 4 has already been verified. To see the planning mechanism more clearly, refer to Figure 5. The first middle-term chain generated connects the DESTINATION assumption to the TRANSPORT goal via premise 23. This is shown by the unifications (deductive interactions) u_1 and u_2 in Figure 5a. The predicate occurrences involving the relations AVAILABLE and OFFLOAD become subproblems. The former is to be given data-base support; the latter is deduced by a middle-term chain from the DAMAGED assumption through premises 7 and 15. This chain is shown in Figure 5b by the unifications u_3, u_4, and u_5. The two new subproblems are to be given data-base support. Thus the plan generated uses three premises and contains three subproblems requiring data-base search. The plausibility of the plan is calculated by a fuzzy intersection (the minimum of the plausibilities of the premises involved--Zadeh [1965]).

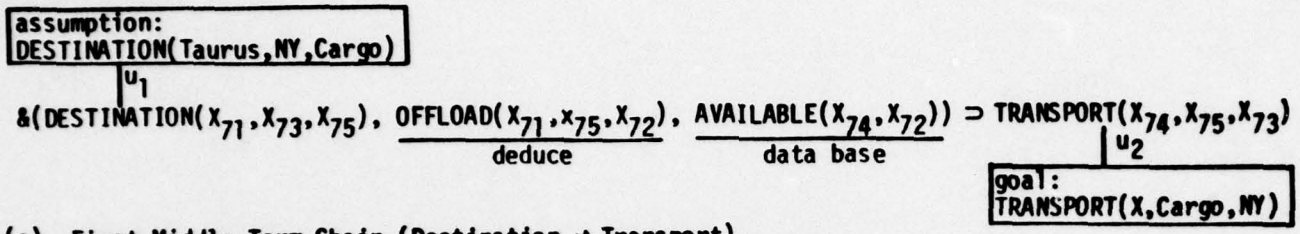
The plan is then verified with variable substitutions inserted in the plan and in the search requests (Figure 4). Note the variable constraints in the search requests. The variable x_{72} represents the home port of Taurus; values found for this variable must be the same as those found for x_{72} in the AVAILABLE search request. Thus, those ships that are available in Taurus's home port are the ones we are interested in. The proof display is given for the first answer found (the Pisces).

In Figure 5b, note that the unifications u_4 and u_5 were computed when these premises were first entered into the system and stored in the PCG. Also stored in the PCG were the truth-functional dependencies within the premises (for example, between DAMAGED and RETURNS, between RETURNS and OFFLOAD, and between DESTINATION and TRANSPORT). The unifications u_1, u_3, and u_2 involve query predicates. Hence they were computed after query input to locate possible middle-term-chain end points. Once these were found, only the PCG was used for middle-term chaining.

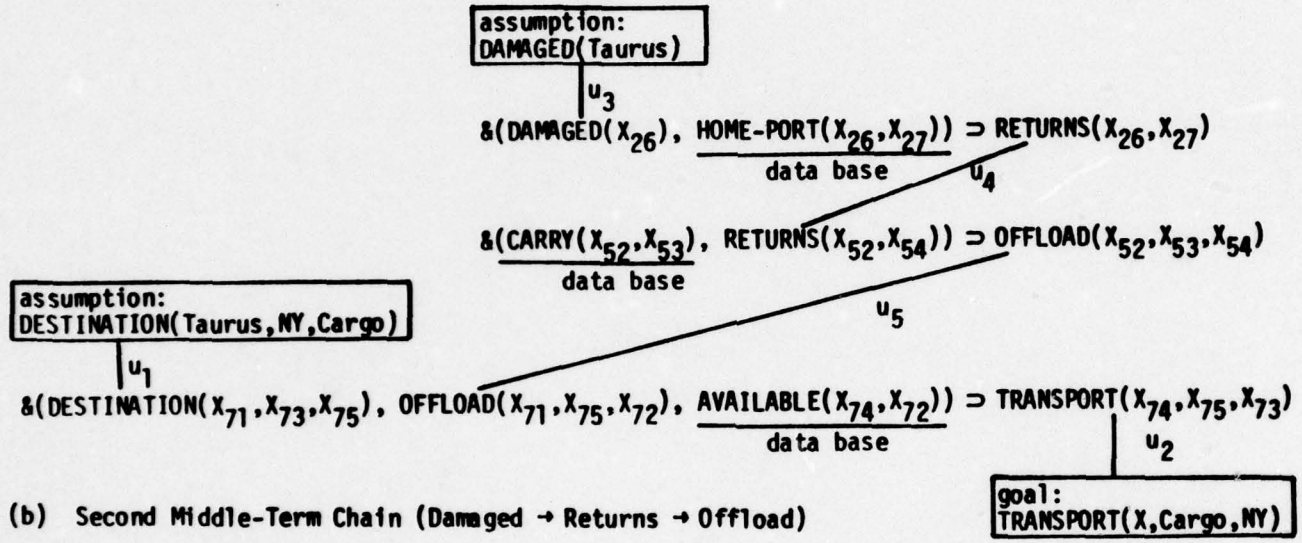
COMPLETENESS ISSUES

The deductive logic on which our system is based is that of an extensional first-order predicate calculus where the issue of logical completeness often arises. In our discussion, we will distinguish between expressional completeness and derivational completeness.

By expressional completeness is meant the ability to represent, in our primitive-conditional form, equivalents of all the well-formed formulas of a first-order predicate calculus. A worry



(a) First Middle-Term Chain (Destination → Transport)



(b) Second Middle-Term Chain (Damaged → Returns → Offload)

Figure 5. Inference Plan Development for Query in Fig. 4.

might arise because only one level of nesting is allowed in primitive conditionals, i.e., the conjuncts or disjuncts of an antecedent or of a consequent must be composed of literals (negated or unnegated predicate occurrences). The worry can be put to rest, however, when it is recalled that even simpler normal forms are expressionally complete, for example, the conjunctive normal form. A conjunctive normal form (CNF) expression is a conjunction of disjunctions of literals. In our logic, a primitive conditional with no antecedents is interpreted as unconditionally asserting the consequent. Thus a CNF disjunction can always be represented as a primitive conditional with a disjunctive consequent and no antecedent; and any CNF expression as a conjunction of such conditionals. Through the use of the inference rules of simplification (ϕ & $\psi \rightarrow \phi$; ϕ & $\psi \rightarrow \psi$) and of adjunction ($\phi, \psi \rightarrow \phi$ & ψ), primitive conditionals may be combined or separated to provide expressional completeness.

By derivational completeness is meant the ability to generate all valid derivations. Our system is derivationally complete in theory, but the important issue for us has been the system's practical efficiency and effectiveness in an applications-oriented environment. That our system is derivationally complete follows from the fact that it is expressionally complete and handles all of the deductive interactions associated with unification (including Skolem functions) as used in resolution systems, as well as all forms of deductive dependencies that may occur between predicates (see Klahr [1975] for more detail). The derivational completeness problem for our system is analogous to the completeness problem for a resolution system constrained to use a set-of-support strategy which has long been known to be derivationally complete (Wos et al. [1965]). Middle-term chains generated in response to a query initially involve the desired conclusion (query goals). Subsequent chains involve subgoals resulting from premises used in chains to query goals, etc.

In practice, almost any performance-oriented planning strategy including ours will initially apply selection constraints that may preclude certain deductive interactions from being considered and thus lead to possible incompleteness. However, successive relaxation of these selection constraints will enable the system to achieve all possible deductive paths.

SUMMARY AND FUTURE PLANS

We have described a deductive processor specifically designed to augment relational data base systems and user-oriented language processors. The processes of deductive pathfinding, inference planning, verification, user review of plans, answer extraction, and proof display have been outlined and illustrated with several examples.

Several of the more important design features that are integral to this approach are:

- Verification (checking for consistency of variable substitutions) and instantiation (data-base search) are delayed until one or more global inference plans have been conconstructed.
- Precomputed deductive interactions (unifications) among premises are used to avoid their constant recomputation during deductive processing.
- Variable types (domain classes) are used to semantically restrict the range of predicate expressions.
- Shortest assumption-to-goal deductive paths are found first.
- Inference plans and data-base access strategies are created from the premise file without requiring access to data-base values.
- Advice can be given on the use of particular premises and predicates to aid in the discovery of relevant inference plans.

The prototype is currently being expanded along several different dimensions in line with our goal of eventually incorporating the deductive processor into an operational data management system and language processor environment. A number of improvements in man-machine interaction and user displays are being made to support more direct and flexible control of plan-generation and data-base search. Additional semantic constraints on the generation of plans will be introduced by expanded use of the semantic network, and by extension of the semantic-advice formalism. We also plan additional investigations in the use of incomplete and plausible knowledge, and logical properties.

ACKNOWLEDGMENTS

The research reported here has been supported by the Advanced Research Projects Agancy of the Department of Defense and is monitored by the Office of Naval Research under Contract N00014-76-C-0885.

REFERENCES

1. Davis, R. and King, J. [1975] An Overview of Production Systems, AIM-271, Artificial Intelligence Laboratory, Stanford University, 1975.

2. Elliott, R. W. [1965] A Model for a Fact Retrieval System, TNN-42, Computation Center, University of Texas, Austin, 1965.

3. Kellogg, C. H., Burger, J., Diller, T. and Fogt, K. [1971] The CONVERSE Natural Language Data Management System: Current Status and Plans, *Proceedings Symposium on Information Storage and Retrieval*, ACM, New York, 1971, 33-46.

4. Kellogg, C., Klahr, P. and Travis, L. [1976] A Deductive Capability for Data Management, In *Systems for Large Data Bases* (P.C. Lockemann and E. J. Neuhold, Eds.), North Holland, Amsterdam, 1976, 181-196.

5. Kellogg, C., Klahr, P. and Travis, L. [1977] Deductive Methods for Large Data Bases, *Fifth International Joint Conference on Artificial Intelligence*, MIT, Cambridge, Mass., 1977, 203-209.

6. Klahr, P. [1975] "The Deductive Pathfinder: Creating Derivation Plans for Inferential Question-Answering," Ph.D. Dissertation, Computer Science Dept., University of Wisconsin, Madison, Wisconsin, 1975.

7. Klahr, P. [1978] Planning Techniques for Rule Selection in Deductive Question-Answering, In *Pattern-Directed Inference Systems* (D. Waterman and F. Hayes-Roth, Eds.), Adademic Press, New York, 1978.

8. Kowalski, R. [1975] A Proof Procedure Using Connection Graphs, *JACM 22*, 4 (October 1975), 572-595.

9. McSkimin, J. and Minker, J. [1978] A Predicate Calculus Based Semantic Network for Question-Answering Systems, In *Associative Networks - The Representation and Use of Knowledge in Computers* (N. Findler, Ed.), Academic Press, New York, 1978.

10. McSkimin, J. and Minker J. [1977] The Use of a Semantic Network in a Deduction Question-Answering System, *Fifth International Joint Conference on Artificial Intelligence*, MIT, Cambridge, Mass., 1977, 50-58.

11. Reiter, R. [1978] Deductive Question-Answering in Relational Data Bases, In *Logic and Data Bases* (H. Gallaire and J. Minker, Eds.), Plenum Press, New York, New York, 1978, 149-177.

12. Robinson, J. A. [1965] A Machine-Oriented Logic Based on the Resolution Principle, *JACM 12*, 1 (January 1965), 23-41.

13. Sickel, S. [1976] A Search Technique for Clause Interconnectivity Graphs, *IEEE Trans. Computers, C-25*, 8 (August 1976), 823-835.

14. Travis, L., Kellogg, C. and Klahr, P. [1973] Inferential Question-Answering: Extending CONVERSE, SP-3679, System Development Corporation, Santa Monica, Calif., 1973.

15. Wos, L., Robinson, G. A. and Carson, D. A. [1965] Efficiency and Completeness of the Set of Support Strategy in Theorem Proving, *JACM 12*, 4 (October 1965), 536-541.

16. Zadeh, L. A. [1965] Fuzzy Sets, *Information and Control 8*, (1965), 338-353.